SQL INJECTION ATTACK – COMPUTER SECURITY

Group 2 :
Yash Rojiwadia
Jay Vora

INTRODUCTION

- SQL Injection Attack is a type of security breach where an attacker injects malicious SQL code into an application's database, exploiting vulnerabilities in the software's handling of user input.
- SQL Injection is a serious threat to the security of any organization that manages sensitive information through web applications.
- This presentation will explore the different types of SQL Injection attacks and provide countermeasures to prevent and mitigate such attacks.

What is SQL Injection?

- A type of security vulnerability that allows an attacker to manipulate SQL queries sent to a database
- Typically involves inserting malicious SQL code into user input fields on a web form or URL parameter
- Can allow an attacker to view, modify, or delete sensitive data in the database

What is SQL Injection Attack?

- SQL Injection Attack is the attack that takes advantage of SQL Injection, a security vulnerability in application or web interface.
- The attack occurs when an application does not validate user input properly before sending it to a SQL database
- Successful SQL Injection attacks can lead to the attacker gaining unauthorized access to sensitive data, such as usernames, passwords, and credit card information

What can SQL Injection Attacks do?

- Retrieve sensitive information :
  - Usernames/ Passwords
  - Credit Card information
- Manipulate Data:
  - Delete records
  - Truncate tables
  - Insert records
- Manipulate Database Objects
  - Drop tables
  - Drop databases

Three Types of SQL Injection Attacks

1. In-band SQL Injection
   - Tautology
   - End of line comment
   - Piggybacked queries

2. Inferential attack
   - Illegal/logically incorrect queries
   - Blind SQL injection

3. Out-of-band SQL Injection

In-band SQL Injection Attacks

1. **Tautology**: An attacker injects malicious code that makes the condition in a query always evaluate to true for a SQL command.
   - Example: SELECT * FROM TABLE1 WHERE ID = 100 OR 1 = 1;

2. **End of line comment**: After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments.
   - Example: SELECT * FROM Table1 WHERE name = ‘bob’ # AND pw = ‘’

3. **Piggybacked queries**: An attacker injects additional queries into the original query to extract data, add or modify data.
   - Example: normal SQL statement + +";"";"+ INSERT (or UPDATE, DELETE, DROP) <rest of injected query>

Live Demo

SQL Injection Attacks

Stopping SQL Injections Attack (Counter-Measures)

Strategies to Stop SQL Injection Attacks

- Check incoming values before executing a query :
  - Check the expected data type, length, and format of incoming values before executing a query
  - For example, if expecting a character value with a length of 2, use a substring with a length of 2 to prevent malicious input
  - Incoming value might only be 1 of x possibilities, so ensure that only valid input is accepted.
  - For example, validate that a parameter is an integer or char(2), depending on the expected format

- Use Parameterized Queries
  - Use parameterized queries or prepared statements to avoid concatenating user input with SQL code.
  - Parameterized queries use placeholders for input values, which are later replaced with sanitized values by the database

- Encrypt URL variable strings
  - Encrypt or hash URL variable strings to prevent attackers from tampering with or intercepting data in transit.

Strategies to Stop SQL Injection Attacks

- Multi-Query:
  - In addition to executing a single SQL statement, some database APIs support executing multiple statements in a single request (multi-query)
  - Multi-query can be useful for improving performance or reducing network overhead, but it can also increase the attack surface for SQL injection

- Example: Multi-Query Injection
  - Consider the following code that executes a multi-query in PHP:

```
$query = "SELECT * FROM users WHERE username='$username'; UPDATE log SET count=count+1 WHERE type='login';";
$result = mysqli_multi_query($conn, $query);
```

- An attacker could use a malicious username like foo'; DROP TABLE users; -- to delete the users table and increment the count in the log table
- Avoid using multi-query: use single-query instead whenever possible to reduce the attack surface of SQL injection

Countermeasures for SQL injection Attack

- **Input Filtering**
- Filter user inputs to stop running the SQL query when detecting malicious injections
- PHP provides filtering APIs to help with input filtering and validation
- Examples of PHP filtering APIs include:
  - filter_has_var(): Checks if variable of specified type exists
  - filter_input_array(): Gets external variables and optionally filters them
  - filter_input(): Gets a specific external variable by name and optionally filters it
  - filter_var(): Filters a variable with a specified filter

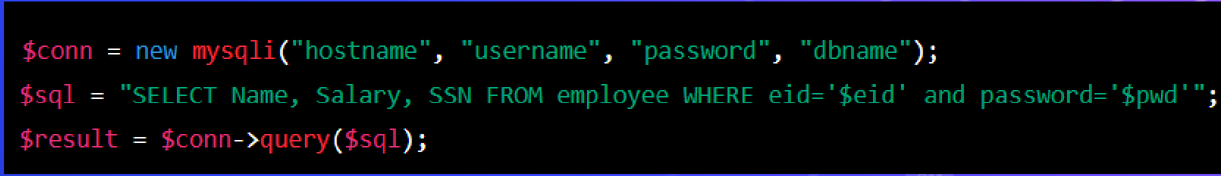
Countermeasures for SQL injection Attack - cont.

Encoding (turning the code into data)

- The apostrophe (‘) is commonly used in SQL injection attacks, if we can encode it, we can prevent the SQL parser from treating it as code.
  - Before encoding:
    - SELECT * FROM employee WHERE eid =‘EID5001’ #’ and password='xyz’
  - After encoding:
    - SELECT * FROM employee WHERE eid =‘EID5001\’ #’ and password='xyz’
- PHP Language APIs:
- $conn = new mysqli("localhost", "root", 'seedubuntu", "dbtest”);
- $eid = $mysqli->real_escape_string($_GET['Username']);
- $pwd = $mysqli->real_escape_string($_GET['Password’]);

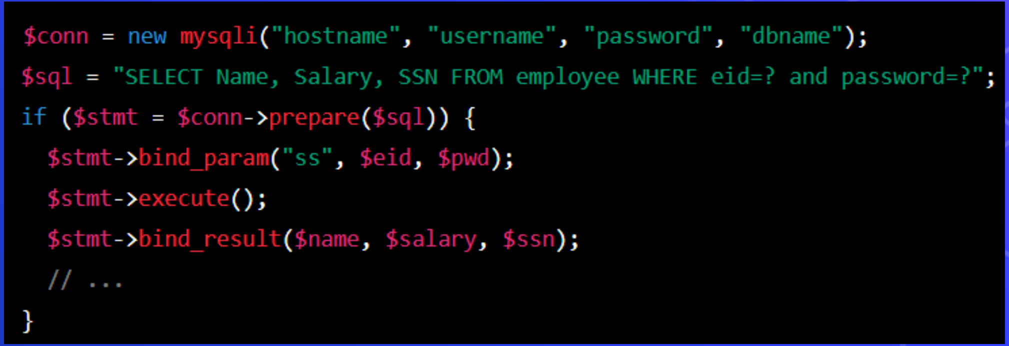
Countermeasures for SQL injection Attack - cont.

- One way to prevent SQL injection attacks is to separate code and data, also known as **parameterization**
- **Parameterization** involves separating the SQL code from the user input, ensuring that user input is never executed as code
- Example: Without Parameterization



```php
$conn = new mysqli("hostname", "username", "password", "dbname");
$sql = "SELECT Name, Salary, SSN FROM employee WHERE eid='$eid' and password='$pwd'";
$result = $conn->query($sql);
```

- This code concatenates user input directly into the SQL query, creating a vulnerability to SQL injection attacks
- Example: With Parameterization



```php
$conn = new mysqli("hostname", "username", "password", "dbname");
$sql = "SELECT Name, Salary, SSN FROM employee WHERE eid=? and password=?";
if ($stmt = $conn->prepare($sql)) {
  $stmt->bind_param("ss", $eid, $pwd);
  $stmt->execute();
  $stmt->bind_result($name, $salary, $ssn);
  // ...
}
```

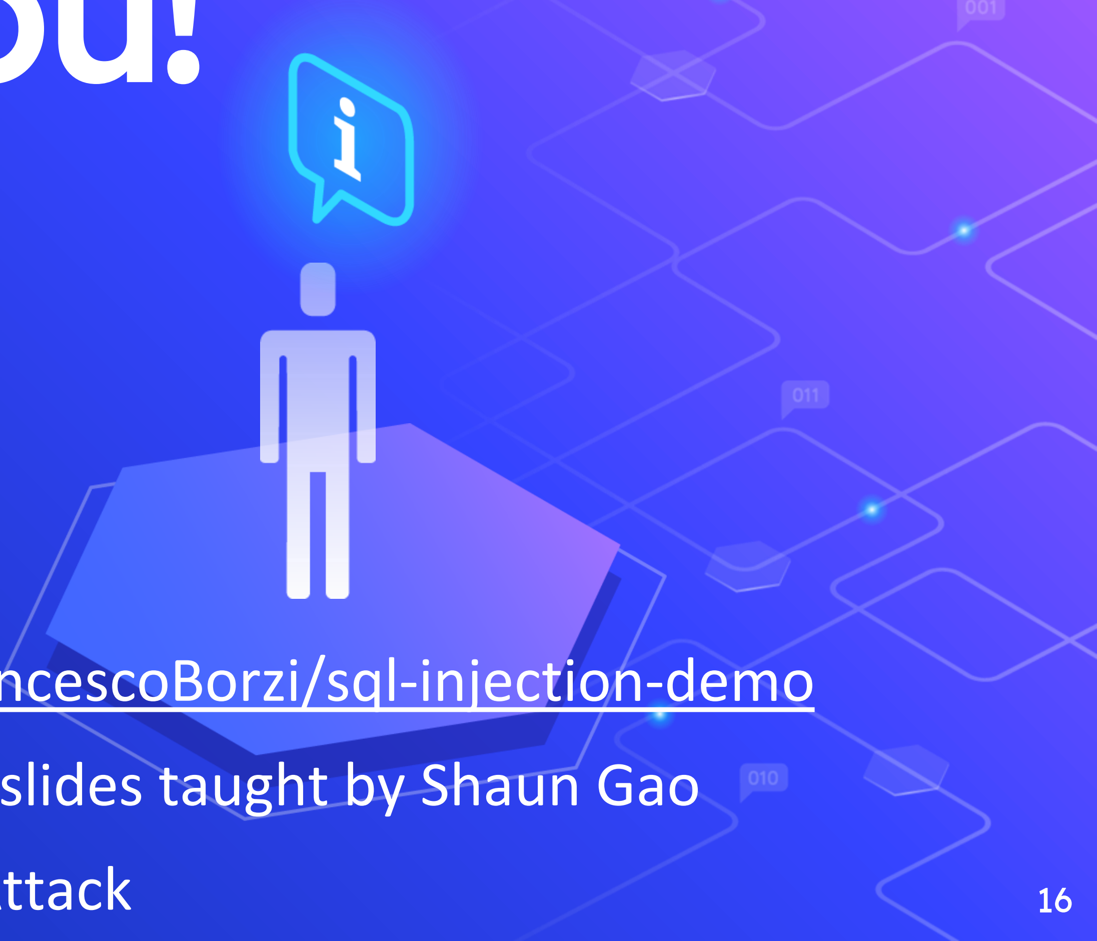
- This code uses placeholders (?) for user input, which are later replaced with sanitized values by the database ensuring that user input is always treated as data, not code

Live Demo

SQL Injections Prevention

Thank you!

Any questions?



Reference:

- Francesco Borzì – GitHub:
  - https://github.com/FrancescoBorzi/sql-injection-demo
- CP400S Computer Security slides taught by Shaun Gao
- Seed Labs – SQL Injection Attack